Конференция научно-технического творчества школьников
«Лабиринты науки»
Секция «Информационно-телекоммуникационные технологии: программирование»

СОЗДАНИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ

Выполнил ученик 11 «Б» класса
ГОУ ЯО «Лицей №86»
Швыров Николай Евгеньевич
Руководитель: учитель информатики
ГОУ ЯО «Лицей №86»
Смирнова Ирина Сергеевна

Ярославль, 2022-2023 учебный год

Содержание

Введение

Компьютеры — это неотъемлемая часть современного мира, одна из самых бурно развивающихся отраслей. Человек смог создать технологию управления машиной, использовать компьютер для решения различных задач. Благодаря языкам программирования процесс управления машиной стал удобным: с их помощью был создан дружественный пользовательский интерфейс, с которым даже далекий от компьютерной техники человек может легко работать.

Но это было не всегда. Создатели и разработчики шли долгим и тернистым путем от неудобных и непонятных машинных кодов до современных высокоуровневых языков программирования, таких как Python, Kotlin, C# и прочих. Благодаря простоте современных языков, их можно изучать и применять, начиная даже со школы.

C++ — один из важнейших и основополагающих языков системного программирования. У C++ есть много недостатков, сложившихся в процессе его долгого исторического развития.

В связи с этим выявлена **проблема**: сложность, неудобство использования и небезопасность языка C++.

Цель данной работы — создать прототип современного языка программирования FatLang, который обладает современными удобными языковыми механиками и решает тот же спектр задач, что и C++, исправив его недостатки.

Задачи данной работы:

1. Изучение синтаксиса и основных механик популярных ЯП;
2. Разработка удобного синтаксиса;
3. Создание транслятора;
4. Оценка проделанной работы.

Методы исследования:

1. Анализ теоретической литературы по изучаемой проблеме;
2. Программирование.

Теоретическая значимость работы заключается в том анализе особенностей довольно часто используемых в коммерческих целях языков.

Практическая значимость: создан новый язык программирования FatLang, привносящий новые возможности.

1. Обзор аналогов

Чтобы создать новое, необходимо изучить что-то уже существующее. Поэтому для создания прототипа современного языка программирования FatLang, отвечающим необходимым требованиям, проведу анализ существующих языков высокого уровня. В обзоре ориентируемся на **критерии подбора аналогов:**

1. Популярность;
2. Прогрессивность решений;
3. Спектр решаемых задач.

В качестве аналогов будут рассмотрены следующие языки: сам C++, Java, D и Rust.

1.1. C++

Язык программирования C++ возник в начале 1980-х годов, когда сотрудник фирмы Bell Labs Бьерн Страуструп придумал ряд усовершенствований к языку C под собственные нужды. Он добавил возможности *объектно-ориентированного* программирования. К 1983 году в язык были добавлены новые возможности, такие как *виртуальные функции*, *перегрузка функций и операторов*, *ссылки*, *константы*. Первый выпуск языка состоялся в октябре 1985 года.

C++ — компилируемый императивный язык общего назначения, на котором можно создавать программы любого уровня сложности. Этот язык поддерживает такие парадигмы, как *объектно-ориентированное*, *структурное*, *процедурное* и *обобщенное* программирование. В процессе работы разработчик получает абсолютную свободу в выборе инструментов для того, чтобы задача, решаемая с помощью того или иного подхода, была

решена максимально эффективно. Иными словами, C++ не принуждает программиста придерживаться только одного стиля разработки программы.

Рисунок 1 — Пример программы “Hello world” на C++

1.2. Java

История создания языка Java начинается в июне 1991 года, когда Джеймс Гослинг создал проект для использования в внутри другого. Oak — первоначальное название Java до 1995 года, затем история этого языка продолжалась под именем Green, а позже — Java. Официальной датой создания данного языка считается 23 мая 1995 года, когда компания Sun

```
#include <iostream>

int main() {
    std::cout << "Hello world" << std::endl;
    return 0;
}
```

выпустила первую реализацию Java 1.0, девиз которой стал: «*Напиши один раз, запускай везде*».

Java — высокоуровневый императивный язык программирования общего назначения. Среда выполнения — виртуальная машина JVM, что немного сказывается на производительности в отрицательную сторону. Достоинство подобного способа выполнения программ — в полной независимости байт-кода от ОС и оборудования, что позволяет выполнять Java-приложения на любом устройстве, которое поддерживает виртуальную машину.

Другой важной особенностью технологии Java является *гибкая система безопасности* благодаря тому, что исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы вызывают немедленное прерывание. Это позволяет пользователям загружать программы, написанные на Java, на их

компьютеры из неизвестных источников, при этом не опасаясь заражения вирусами, пропажи ценной информации, и т. п.

В отличии от C++, язык программирования Java целиком погружает разработчика в мир ООП. Java имеет высокую надежность, что позволяет использовать его в банковских системах и даже при написании операционных систем, примером может послужить JavaOS.

```
package com.program.Main;

public class Main {
    public static void main(String[] args) {
        Sustem.out.println("Hello world");
    }
}
```

Рисунок 2 — Пример программы “Hello world” на Java

1.3. D

История языка программирования *Mars* (первое название данного языка) берет свое начало в 2001 году. Первоначально этот проект задумывался Брайтом как реинжиниринг языка C++ с целью избавиться от наиболее существенных недостатков исходного языка и внедрить в него современные архитектурные решения. При его создании была сделана попытка соединить *производительность* компилируемых языков программирования с *безопасностью* и *выразительностью* динамических.

D — язык программирования, который помогает программистам справиться с современными проблемами разработки ПО. Он разработан с учетом программирования на системном уровне и привносит современный языковой дизайн с простым синтаксисом, подобным C. По словами Андрея Александреску, этот язык программирования создает все условия для организации взаимодействия модулей через *точные интерфейсы*, поддерживает целую федерацию тесно взаимосвязанных парадигм программирования (*императивное*, *объектно-ориентированное*,

функциональное и *метапрограммирование*), обеспечивает *изоляцию потоков*, модульную *безопасность типов*, предоставляет *рациональную модель памяти* и многое другое.

```
import std.stdio;
void main()
{
    writeln("Hello world");
}
```

Рисунок 3 — Пример программы “Hello world” на D

1.4. Rust

История языка программирования Rust начинается в 2006 году, когда Грейдон Хоар решил создать язык с высоким уровнем надежности. В 2009 году Mozilla начала спонсировала этот проект, а в 2010 взяла разработку в свои руки.

Rust — молодой компилируемый императивный язык общего назначения. В Rust поддерживаются *функциональное*, *параллельное*, *процедурное* программирование, а также присутствуют отголоски *ООП* — почти весь спектр реально используемых в прикладном программировании парадигм. Данный язык ориентирован на *безопасность*, *скорость* и *параллелизм*, он часто рассматривается как перспективный язык для разработки ядер операционных систем.

Данный язык позиционирует себя как альтернатива C/C++, что уже само по себе интересно, так как претендентов на конкуренцию не очень много: разве что вышеупомянутый D и Go от Google. С 2016 по 2020 год Rust занимал первое место в списке «Most loved programming languages» по версии ежегодного опроса разработчиков Stack Overflow Developer Survey.

Большой проблемой Rust является неудобный синтаксис, который в некоторых ситуациях может быть довольно сложно прочитать.

```
fn main() {
    println!("Hello world");
}
```

Рисунок 4 — Пример программы “Hello world” на Rust

1.5. Выводы

Итак, каждый из представленных языков программирования может быть использован для решения различного спектра задач, начиная с написания операционных систем. Java и D можно рассматривать как языки с образцовым синтаксисом, Rust привносит современные и довольно интересные решения в программирование, а C++ — образец сочетания высоко- и низкоуровневого программирования. Подробную таблицу сравнения приведенных выше языков можно посмотреть в приложении №3.

2. Процесс создания языка программирования FatLang

2.1. Требования к создаваемому языку FatLang

Ориентируясь на анализ языков-аналоговов, представленный в главе 1 и их сравнительную характеристику (приложение №3), представим требования, которым должен отвечать создаваемый язык FatLang.

1. **Система пакетов и пакетный менеджер.** Одной из наиболее главных проблем C++ является отсутствие системы пакетов. В настоящий момент в языке существует только импорт заголовочных файлов целиком. В случае, когда два таких файла зависят друг от друга, компилятор выдаст ошибку о циклическом импорте. Система пакетов же предполагает циклические зависимости между файлами, а также предполагает неполный импорт пакетов, что, несомненно, является большим преимуществом.

2. **Безопасное программирование.** Языки системного программирования обычно располагают большим и разнообразным инструментарием для решения всевозможных задач. Например, указатели, арифметика указателей, ручной контроль памяти являются изначально небезопасными. C++ никак не контролирует их использование, из-за чего могут происходить трудновыявляемые и трудно исправляемые ошибки. В языке Rust есть интересный языковой механизм *safe* и *unsafe* подмножеств. Благодаря ему разработчик может помечать небезопасные функции и явно указывать места с небезопасным кодом. Это значительно облегчает исправление неожиданных ошибок *segmentation fault* и почти полностью их предотвращает.

3. **Современные языковые особенности.** Языки Rust и D располагают множеством различных особенностей, удобств, значительно облегчающих написание и понимание кода. Например, в C++ сильнее недостает современных enum'ов с возможностью "вкладывать" в них значения и pattern matching'a таких enum'ов с применением ключевого слова *match*.

Также удобной особенностью могут являться *свойства*, позволяющие улучшить читаемость кода.

4. **Поддержка внешнего нативного кода.** Это ключевой механизм всех языков программирования, позволяющий использовать скомпилированный нативный код для фундаментального расширения возможностей.

2.2. Транслятор языка программирования FatLang

Продуктом данного проекта является программа-транслятор. Выбирая язык программирования для написания транслятора, я остановил свой выбор на Python. Основные преимущества данного выбора — довольно быстрое прототипирование, удобная работа со строками и наличие библиотеки для синтаксического анализа PLY. В моем продукте можно выделить следующие ключевые части: *лексер*, *парсер*, *статический анализатор* и *генератор C-кода*. Лексер и парсер реализованы исключительно с помощью библиотеки PLY, в то время как статический анализатор и генератор кода написаны с нуля.

3. Краткий обзор языка FatLang

3.1. Базовые типы данных

Все типы данных можно разделить на две группы: простые (примитивы) и составные. Простые типы — основные элементы построения программы. Это, например, целые числа. Язык FatLang предоставляет следующие примитивы: *i32*, *i64*, *ui32*, *ui64*, *usize (целочисленные)*, *f32*, *f64 (типы с плавающей точкой)*, *bool (логический тип)*, *char (символьный тип)*. Цифра в названии означает сколько бит памяти будет задействовано при использовании данного типа. Приставка *u- (unsigned)* означает, что данный целочисленный тип будет беззнаковым. Длина тип *usize* зависит от разрядности целевой системы (32 или 64 бита). Составные типы — типы, построенные на примитивах. Это массивы, списки, словари, пользовательские классы и т.п.

3.2. Базовые операторы

Язык программирования FatLang предоставляет довольно большой список базовых операторов. В приложении №1 приведены их языковые реализации. В колонке “Базовая типовая реализация” показано то, с какими типами может быть использован оператор, и через тире указан (возвращаемый) тип операции.

Язык соблюдает общепринятые обозначения операторов для удобства пользователя. Логические операторы *&&*, *||* были упрощены до *&*, *|* соответственно. Так как язык строго типизирован, данное упрощение помогает немного повысить скорость написания и значительно повысить читаемость кода.

3.3. Основные синтаксические конструкции

Язык программирования FatLang обладает довольно большим списком синтаксических конструкций. Основные — *if*, *if-else*, *while*, *for*, *struct-declaration (объявление структуры)* и *function-declaration (объявление*

функции). match-statement, enum Именно на эти конструкции пользователь будет опираться при разработке программ.

Синтаксис конструкций (рис. 5-8) *if*, *while*, *struct-declaration* и *function-declaration* был сделан таким, чтобы быть максимально похожим на синтаксис C-подобных языков. Это облегчит процесс освоения языка пользователем.

```
if !false == true {
    //body..
}
else {

}
```

Рисунок 5 — синтаксис if

```
while true {
    //do something..
}
```

Рисунок 6 — синтаксис while

```
struct A {
    i32 a;
}

struct B<T> {
    T val;
}
```

Рисунок 7 — синтаксис структур

```
i32 example() {
    return 1;
}

void example<T>(T arg) {
    //body..
}
```

Рисунок 8 — синтаксис функций

3.4. Пакеты

Система пакетов — мощный инструмент при разработке крупной программы. Благодаря ей возможны циклические зависимости между

файлами, каждый класс и каждая функция имеет свое полное уникальное имя, что расширяет полиморфические возможности языка (можно создать функции с одинаковыми названиями в разных пакетах).

Концепция пакетов позаимствована из языка Java:

1. Каждый файл кода должен начинаться с конструкции “*package [name]”*;
2. Возможно обратиться к любому классу и функции через полное имя;
3. Есть возможность импорта пространств имен с помощью конструкции “*using [name]”*.

3.5. Простейшая программа

На рисунке 5 показана программа “Helloworld”. В первой строчке обязательно обозначить пакет, с которым связан данный файл. Второй строчкой идет подключение пакета *io* из стандартной языковой библиотеки.

Строки 4-7 занимает функция *main* — стартовая точка программы. В строке 5 используется функция пакета *std:io println* для вывода строки “Hello world”. Затем из функции возвращается 0 — код выхода из программы (как в C/C++).

Рисунок 9 — Простейшая программа “Helloworld”

4. Экономика

Существует два способа оценки стоимости программы: по затраченному времени (почасовая) и по количеству строк в проекте (построчная). Оценить время написания большого проекта, не имея опыта, — сложная задача. Также стоит учитывать и школьную нагрузку. Для построчной оплаты достаточно просто посчитать количество строк во всем коде.

Я бы оценил время создания транслятора в *3 месяца*. Учитывая школьную и внешкольную нагрузку я бы смог работать от *2 до 4 часов* в день над проектом. Возьмем среднее — 3 часа в день. Получается около *270 часов*. Ставки за час работы могут варьироваться от компании к компании и зависят от уровня программиста, поэтому будет отталкиваться от фрилансового рынка. Новичку со ставкой 100 рублей/час никто бы не доверил такой большой проект, поэтому возьмем среднего опытного программиста, который будет брать *450 рублей за час* работы. То есть в итоге выходит *121,5 тыс. рублей*.

Оценивая построчно получаются совсем иные результаты. В моём проекте около 5000 строк кода. Также отталкиваясь от фрилансового рынка, за 1 строку кода Python-разработчика цена 1-3 рубля. Среднее — 2 рубля за строку. В итоге получается *10 тыс. рублей*.

Мне кажется почасовая оценка времени работы будет здесь более корректной: большой и сложный проект требует тщательного обдумывания, много времени будет уходить просто на разработку архитектуры программы.

Продуктом данного проекта является бесплатная *open-source* программа, что не предполагает коммерческого использования.

Список источников

1. Язык программирования D [Электронный ресурс] — Режим доступа: https://dlang.org/
2. Язык программирования Rust [Электронный ресурс] — Режим доступа: https://www.rust-lang.org/
3. Буйначев, С. К. Основы программирования на языке Python : учебное пособие / С. К. Буйначев, Н. Ю. Боклаг. – Екатеринбург : Изд-во Урал. ун-та, 2014. – 91, [1] с.
4. Рейзлин В.И. Язык C++ и программирование на нём: учебное пособие / В.И. Рейзлин; Томский политехнический университет. – 3-е изд., перераб. – Томск : Изд-во Томского политехнического университе-та, 2021. – 208 с.
5. А. В. Гаврилов, О.А. Декиярёва, И.А. Лёзин, ИВ. Лёзина, УЧЕБНОЕ ПОСОБИЕ ПО ЯЗЫКУ JAVA (Часть 1), Самара [Электронное учебное пособие] — Режим доступа: http://repo.ssau.ru/bitstream/Uchebnye-posobiya/Uchebnoe-posobie-po-yazyku-Java-elektron-ucheb-posobie-Ch-1-54324

Приложение №1

Базовые операторы

Оператор	Базовая типовая реализация	Описание
+	*i32* + *i32* — *i32* *ui32* + *ui32* — *ui32* … *f32* + *f32* — *f32* *f64* + *f64* — *f64*	Сложение
- *(бинарный)*	*i32* - *i32* — *i32* *ui32* - *ui32* — *ui32* … *f32* - *f32* — *f32* *f64* - *f64* — *f64*	Вычитание
*	*i32* * *i32* — *i32* *ui32* * *ui32* — *ui32* … *f32* * *f32* — *f32* *f64* * *f64* — *f64*	Умножение
/	*i32* / *i32* — *i32* *ui32* / *ui32* — *ui32* … *f32* / *f32* — *f32* *f64* / *f64* — *f64*	Деление
%	*i32* % *i32* — *i32* *ui32* % *ui32* — *ui32* … *usize* % *usize* — *usize*	Остаток от деления
&	*i32* & *i32* — *i32* *ui32* & *ui32* — *ui32* … *usize* & *usize* — *usize*	Побитовое “И”
&	*bool* & *bool* — *bool*	Логическое “И”
|	*i32* | *i32* — *i32* *ui32* | *ui32* — *ui32* … *usize* | *usize* — *usize*	Побитовое “ИЛИ”
|	*bool* | *bool* — *bool*	Логическое “ИЛИ”
^	*i32* ^ *i32* — *i32* *ui32* ^ *ui32* — *ui32* …	Побитовое “Исключающее ИЛИ”

	usize ^ usize — usize	
^	*bool ^ bool — bool*	Логическое “Исключающее ИЛИ”
<<	*i32 << i32 — i32* *ui32 << ui32 — ui32* … *usize << usize — usize*	Сдвиг влево
>>	*i32 >> i32 — i32* *ui32 >> ui32 — ui32* … *usize >> usize — usize*	Сдвиг вправо
<	*i32 < i32 — bool* *ui32 < ui32 — bool* … *f32 < f32 — bool* *f64 < f64 — bool*	Меньше
>	*i32 > i32 — bool* *ui32 > ui32 — bool* … *f32 > f32 — bool* *f64 > f64 — bool*	Больше
<=	*i32 <= i32 — bool* *ui32 <= ui32 — bool* … *f32 <= f32 — bool* *f64 <= f64 — bool*	Меньше или равно
>=	*i32 >= i32 — bool* *ui32 >= ui32 — bool* … *f32 >= f32 — bool* *f64 >= f64 — bool*	Больше или равно
==	*i32 == i32 — bool* *ui32 == ui32 — bool* … *f32 == f32 — bool* *f64 == f64 — bool* *bool == bool — bool*	Равно
!=	*i32 != i32 — bool* *ui32 != ui32 — bool* … *f32 != f32 — bool* *f64 != f64 — bool* *bool != bool — bool*	Не равно

- *(унарный)*	*-i32 — i32* *-i64 — i64* *-f32 — f32* *-f64 — f64*	Изменение знака на противоположный
!	!*i32 — i32* !u*i32 — ui32* ... !*usize — usize*	Побитовое “НЕ”
!	!*bool — bool*	Логическое “НЕ”

Приложение №2

Программа для решения задачи ЕГЭ по информатике

На вход алгоритма подаётся натуральное число *N*. Алгоритм строит по нему новое число *R* следующим образом.
1) Строится двоичная запись числа *N*.
2) К этой записи дописываются справа ещё два разряда по следующему правилу:
а) складываются все цифры двоичной записи, и остаток от деления суммы на 2 дописывается в конец числа (справа). Например, запись 11100 преобразуется в запись 111001;
б) над этой записью производятся те же действия — справа дописывается остаток от деления суммы цифр на 2.
Полученная таким образом запись (в ней на два разряда больше, чем в записи исходного числа *N*) является двоичной записью искомого числа *R*.
Укажите минимальное число *R*, которое превышает 43 и может являться результатом работы алгоритма. В ответе это число запишите в десятичной системе.

```
package main;

using std;
using std:io;

i32 sumBits(i32 n) {
    i32 s = 0;
    while (n > 0) {
        s += n & 0b1;
        n >>= 1;
    }
    return s;
}

i32 f(i32 n) {
    n = n << 1 | sumBits(n) % 2;
    n = n << 1 | sumBits(n) % 2;
    return n;
}

i32 main() {
    i32 i = 0;

    while i < 1000 {
        i32 r = f(i);
        if r > 43 {
            println(i);
            println(r);
            return 0;
        }
        i += 1;
    }

    return 0;
}
```

Приложение №3

Критерий	C++	Java	Rust	D
Направленность языка	Общая	Общая	Общая	Общая
Современность языка	Устаревший	Устаревший	Современный	Современный
Скорость разработки	Небольшая	Средняя	Небольшая	Средняя
Скорость изучения	Очень медленная	Медленная	Медленная	Медленная
Среда выполнения	Нативно	JVM	Нативно	Нативно
Организация взаимодействия файлов проекта	Прямое “включение” файлов через include	Пакеты	Модули	Модули
Безопасное программирование	-	+	+	+
Управление памятью	Ручное	Сборщик мусора	Ручное, частично автоматическое	Сборщик мусора
Наличие препроцессорных максоров	+	-	+	-